



Nuestro compromiso es con el *futuro*.

# Back End

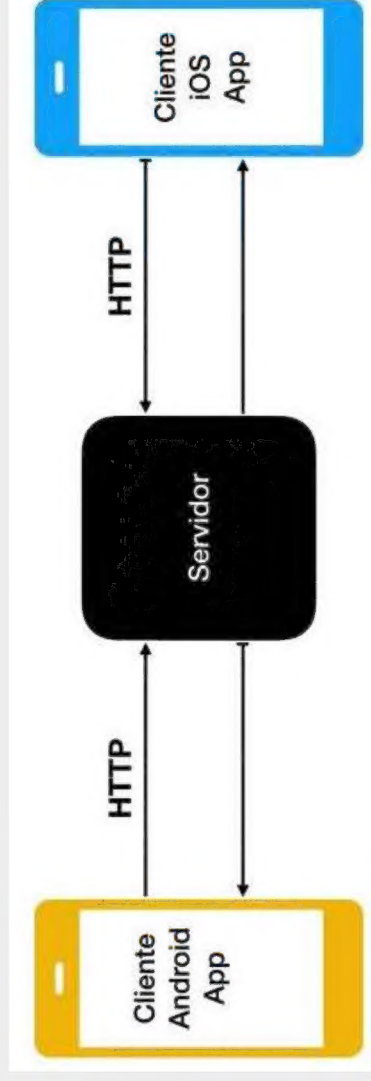
# Clase 9

# ¿Qué veremos hoy?

Hoy vamos a continuar con el módulo de **Back-End**, donde veremos el concepto de **API REST**, y que cliente podemos utilizar para poder invocarla. Dentro de este apartado, propondremos como cliente **API REST** a **insomnia**. Relacionado al concepto de **API REST** veremos como hacer uso de las librerías que hemos venido utilizando hasta el momento (express, ejs, etc) para poder definir nuestras propias API's y así también consumir API's externas.

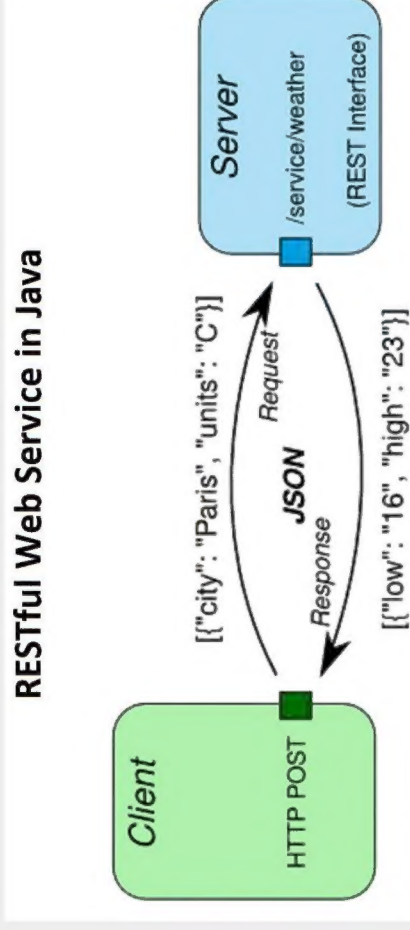
# Introducción al concepto de API

La palabra **API** es un acrónimo que significa Interfaz de Programación de Aplicaciones (Application Programming Interface). Es un sistema que funciona como intermediario entre diferentes aplicaciones de software y su función es permitir que estas aplicaciones puedan comunicarse entre sí.



# Termino 'REST'

**REST** es un conjunto de principios que definen la forma en que se deben crear, leer, actualizar y eliminar los datos. Es una arquitectura conocida como cliente-servidor, en la que el servidor y el cliente actúan de forma independiente, siempre y cuando la interfaz sea la misma al procesar una solicitud y una respuesta, qué son los elementos esenciales. El servidor expone la **API REST** y el cliente hace uso de ella. El servidor almacena la información y la pone a disposición del usuario, mientras que el cliente toma la información y la muestra al usuario o la utiliza para realizar posteriores peticiones de más información.



# API REST Verbos HTTP

**Http** nos ofrece los siguientes verbos, los cuales son utilizados y hemos venido utilizando hasta el momento, mediante el uso de expres. Estos son:

## **GET**

El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

## **HEAD**

El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

## **POST**

El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

## **PUT**

El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

## **DELETE**

El método DELETE borra un recurso en específico.



# API REST Códigos HTTP

## Respuesta

Cada petición **HTTP** tiene un código de respuesta asociado, los cuales son:

### **200 OK**

La solicitud ha tenido éxito. El significado de un éxito varía dependiendo del método HTTP:

### **201 Created**

La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello. Ésta es típicamente la respuesta enviada después de una petición **PUT**.

### **400 Bad Request**

Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.

### **401 Unauthorized**

Es necesario autenticar para obtener la respuesta solicitada. Esta es similar a 403, pero en este caso, la autenticación es posible.



# Creando nuestra primera API

Para crear nuestra **API** utilizaremos **Express**, que es un framework de **Nodejs**.

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hola Mundo!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Express **JS**

Si nos dirigimos a <http://localhost:3000/> veremos como respuesta **"Hola Mundo!"**. A continuación veremos qué cliente podemos usar para invocar a nuestra API.

# API Client 'Insomnia'

**Insomnia** es un cliente **REST** multiplataforma, con una interfaz clara y sencilla. Con funcionalidades que nos van a facilitar enormemente el trabajo. Para descargar al mismo nos dirigimos a <https://insomnia.rest/download>.

# Usando 'Insomnia'

Una vez instalado el cliente podríamos comenzar a usar el cliente para invocar al servicio que hicimos en la filmina anterior, el que nos devolvía "Hola Mundo!"



Se aprecia la interfaz que nos da Insomnia y nuestra primer petición al servicio mencionado. Como vimos antes, se observa también el código de respuesta, que en esta caso es **HTTP 200 OK**.

# Sincronismo VS Asincronismo

Un código **síncrono** es aquel código donde cada instrucción espera a la anterior para ejecutarse mientras que un código **asíncrono** no espera a las instrucciones diferidas y continúa con su ejecución. Por lo general la asincronía permite tener una mejor respuesta en las aplicaciones y reduce el tiempo de espera del cliente.

Dentro del ecosistema de **JS**, disponemos de **fetch**, con la cuál podemos realizar peticiones **HTTP** asíncronas utilizando **promesas** y de forma que el código sea un poco más sencillo y menos verbose. Pero qué entendemos por **promesa**?

Una **Promise** (promesa en castellano) es un objeto que representa la terminación o el fracaso de una operación asíncrona. Dado que la mayoría de las personas consumen promises ya creadas, esta guía explicará primero cómo consumirlas, y luego cómo crearlas.

# Ejemplo 'Fetch'

```
fetch('http://example.com/movies.json')  
  .then(response => response.json())  
  .then(data => console.log(data));
```

1. Hemos llamado a **fetch()** con la URL a la que queremos acceder como parámetro. Esta llamada nos devuelve una promesa.
2. El método **then()** de esa promesa nos entrega un objeto response.  
A partir de este objeto llamamos al método **json()** para obtener el cuerpo de la respuesta. Este método **json()** nos devuelve otra promesa que se resolverá cuando se haya obtenido el contenido.
3. El método **then()** de esta segunda promesa recibe el cuerpo devuelto por la promesa anterior y hace un log de ella.

**¡Vamos al código!**



**¡No olvidemos concurrir a clases de  
consulta!!**

# Muchas gracias!



ICARO Asociación Civil

CUIT 30716564815

[info@icaro.org.ar](mailto:info@icaro.org.ar)

[www.icaro.org.ar](http://www.icaro.org.ar)